

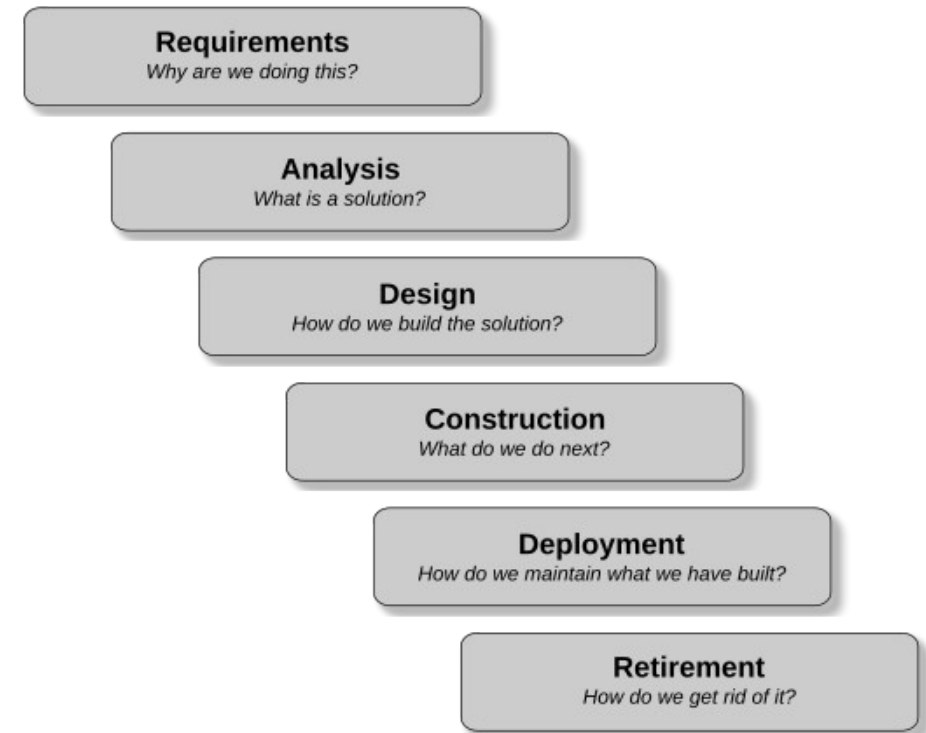


# Software Engineering

## 2: Process Types

# The Engineering Cycle

- The Engineering Cycle is a set of logical steps
  - Each step builds on the previous one
  - How we apply this cycle is a process type.
- Waterfall Process
  - Characterized by completing each step in the process before moving on to the next.
  - Also called a predictive process because given the full set of requirements and technical constraints, we can accurately predict the outcome.
  - Common in engineering systems and high risk systems
    - Like nuclear reactor control software or airline navigation software
  - Or where requirements and technology doesn't change over the lifetime of the project



- For a lot of applications, the waterfall doesn't work
  - Due to uncertainty and variation at each stage of the engineering cycle
  - Can create a lot of rework
- Manufacturing started to experiment in 1950s-1980s
  - Very influential innovation: Lean manufacturing & Toyota production system
- In the 1980s, there was a major crisis in software development
  - At the time, most software was being developed waterfall (big bang) style
  - Siloed teams (design, development, testing) with one-time hand offs of artifacts
  - Same issues addressed in the NATO 1969 conference
  - Recommended adopting the lean, iterative and incremental approaches being used in manufacturing

# Adaptive Methodologies

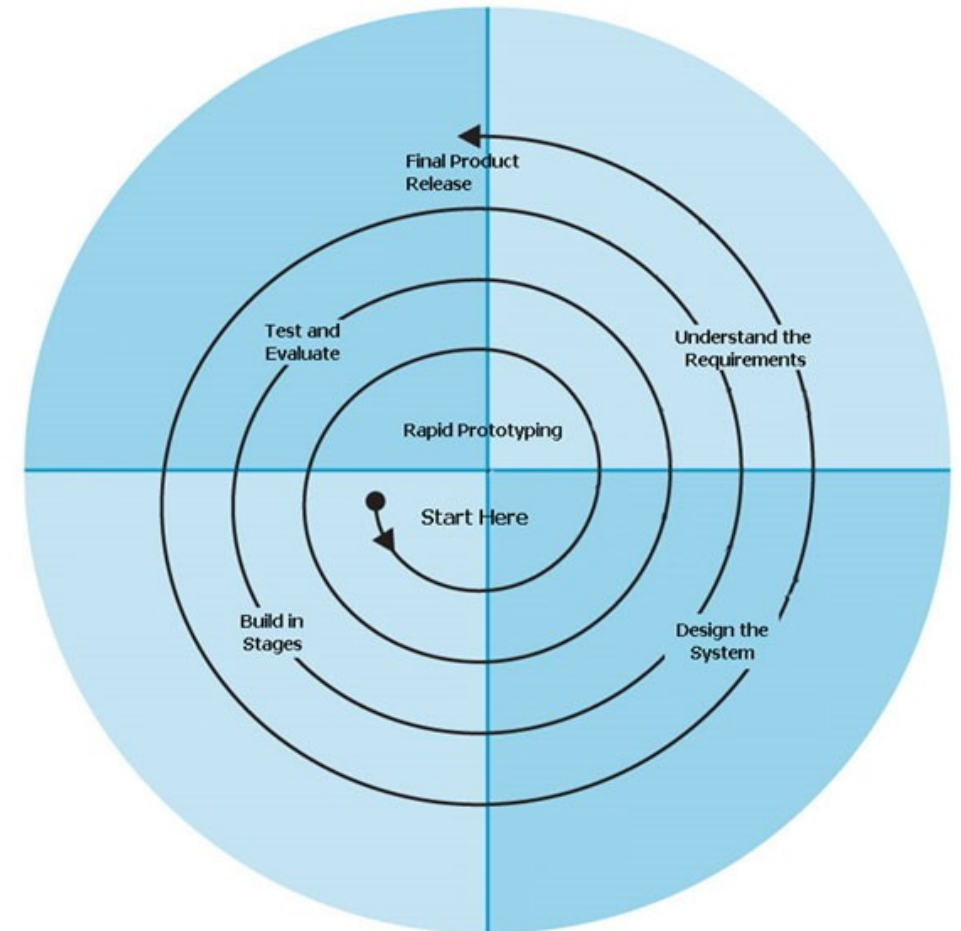
- In the 1980s and 1990s
  - Companies experimented with using what they called adaptive approaches
  - IBM, DuPont, and others experimented with iterative prototyping and empirical process control.
  - Derived from the work mentioned in the previous slide and others (especially lean)
- Characterized by
  - Use of successive prototypes to get feedback on requirements, design and performance
  - Short iterations (one month or less),
  - Cross-functional teams,
  - Daily meetings for synchronization,
  - A prioritized feature list

# SCRUM

- The term Scrum is first used in 1986 with reference to new product development
  - Hirotaka Takeuchi and Ikujiro Nonaka “The New New Product Development Game”
  - Not originally developed for software
  - Jeff Sutherland & Ken Schwaber (1997) Presented the first public paper on Scrum: "SCRUM Development Process" adapting it to software development
- Emphasized:
  - Empirical process control theory (transparency, inspection, adaptation),
  - Lean principles,
  - Nonaka & Takeuchi’s knowledge-creation theory (SECI) – the basis for cross functional teams
- The basic ideas expressed in Scrum started to be adopted
  - Primarily by teams working with highly adaptive types of projects
  - Often smaller teams of developers working closely with the business side

# Non-Agile Adaptive Methodologies

- Influences on Agile were some of the adaptive methods used in computer engineering
  - Barry Boehm's Spiral methodology from 1986 defined a series of iterations with the objective of producing prototypes which were used to provide inputs into the subsequent iterations.
  - Another adaptive methodology is James Martin's Rapid Application Development (RAD) developed in the 1980s at IBM.
- Both SDLCs was built around the idea that for some sorts of development, like working with user interfaces, the requirements are too fluid for a predictive approach.
- The RAD approach, like the Spiral methodology, centered around getting a prototype into the hands of the users to start generating feedback that would be used to continuously develop the product



<http://softwaretesting-qaqc.blogspot.com>

# Scrum and Agile

- Agile was the general term adopted in 2001
  - By a consortium of owners of adaptive methodologies that shared a similar approach to development derived from their use of Scrum ideas
  - eg. Extreme Programming, Feature Driven Development
  - As a result, many of the features of Scrum were incorporated into the Agile Manifesto and the Agile Principles
- Note that Scrum is NOT an Agile methodology, but its concepts were shared among Agile methodology
  - Specifically:
    - Individuals and interactions over formal documentation
    - Working software prototypes
    - Customer collaboration and feedback loops
    - Responding to change.



# Agile Principles

- The various Agile methodologies summarized their approach to building software through a statement of Agile principles:
  - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - We welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
  - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
  - Business people and developers must work together daily throughout the project.
  - Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
  - The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
  - Working software is the primary measure of progress.
  - Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace



# Agile Principles

- Continuous attention to technical excellence and good design enhances agility.
  - Simplicity--the art of maximizing the amount of work not done--is essential.
  - The best architectures, requirements, and designs emerge from self-organizing teams.
  - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
- Note that Agile principles are about how we organize our development process
    - It does NOT address programming issues or how we write our code
    - The is up to the individual methodology, for example, Extreme Programming (XP)

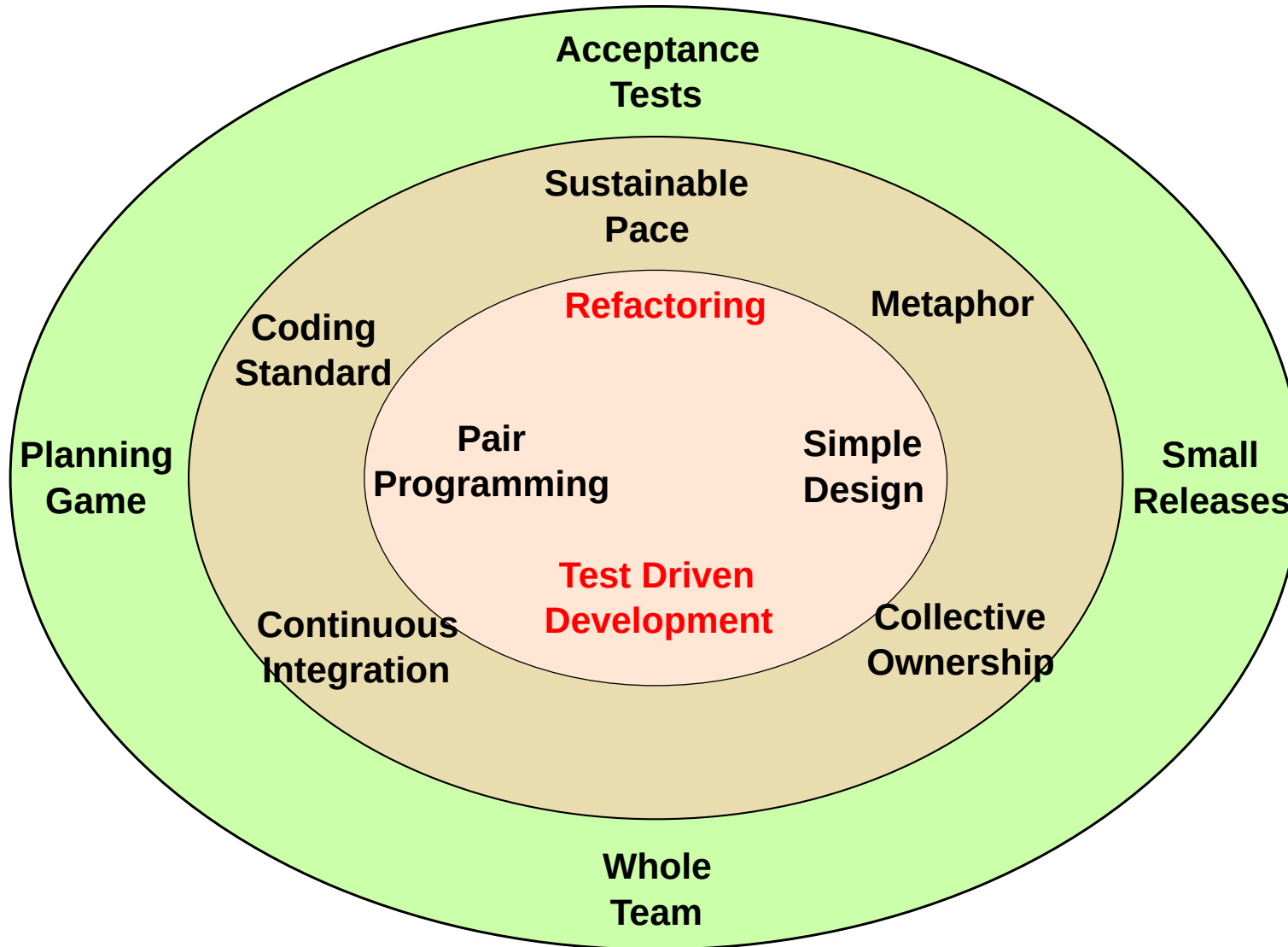
# Extreme Programming (XP)

- Example of a development methodology that defines the actual programming practices to complement the process
- Many modern programming techniques originated in a methodology called Extreme Programming or XP
  - Created by Kent Beck.
  - XP is extreme means taking 12 development "best practices" to their logical extremes
- XP is intended to be easily used for projects of up to a dozen programmers and twice that with some difficulty
  - XP itself does not scale well
  - The way to do large scale XP development is within a project organized overall along more traditional models, but is then split into multiple smaller XP projects

# Extreme Programming (XP)

- Example of a development methodology that defines the actual programming practices to complement the process
- Many modern programming techniques originated in a methodology called Extreme Programming or XP
  - Created by Kent Beck.
  - Saying XP is extreme means taking 12 development "best practices" to their logical extremes
- Organized the specific activities to be used in programming
  - Designed so that the programming practice could easily map into the development process
- Organized these into the XP onion
  - Many have these have been adopted as software engineering best practices

# Xp Onion



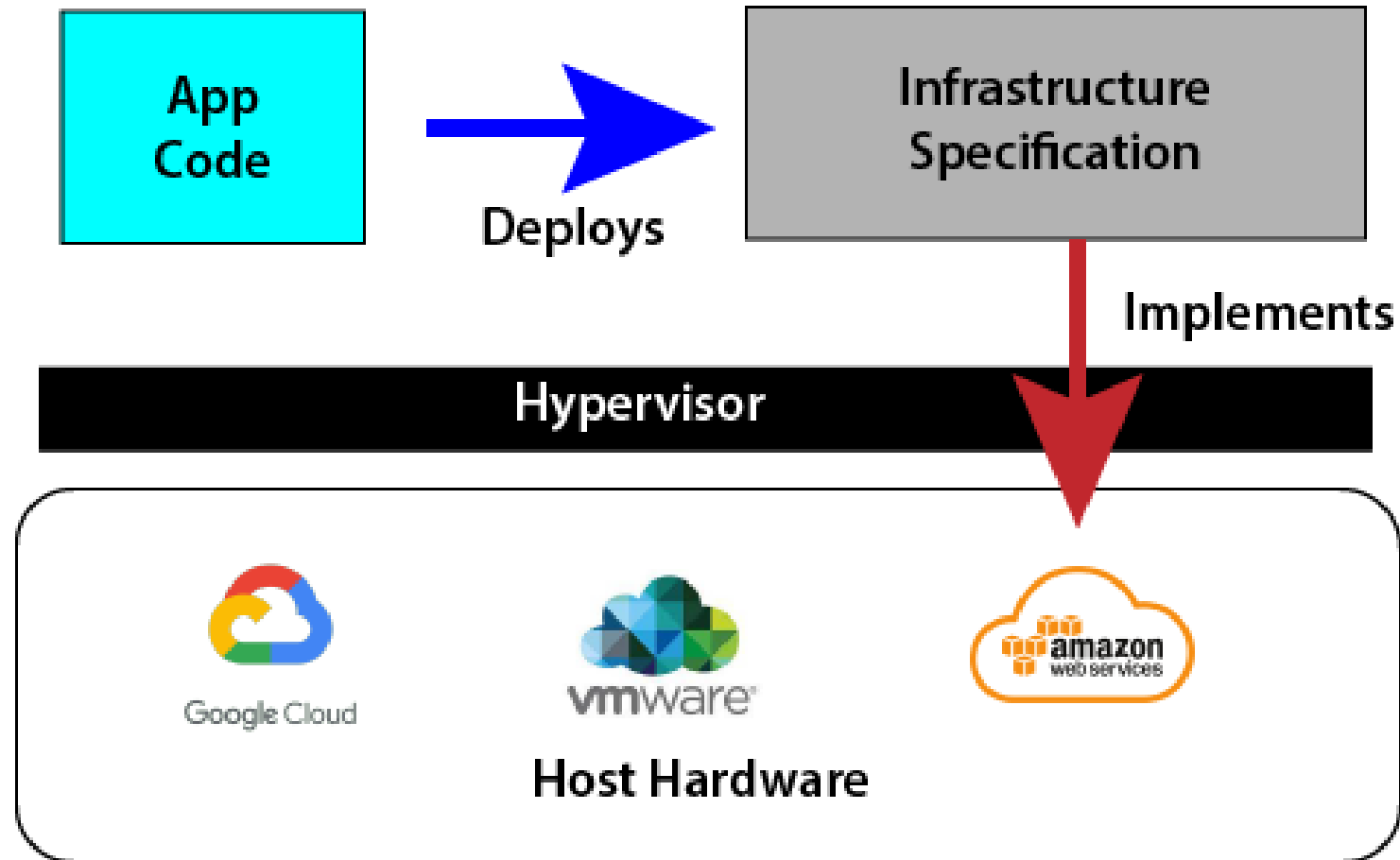
# Extreme Programming (XP)

- Example of a development methodology that defines the actual programming practices to complement the process
- Many modern programming techniques originated in a methodology called Extreme Programming or XP
  - Created by Kent Beck.
  - Saying XP is extreme means taking 12 development "best practices" to their logical extremes
- Organized the specific activities to be used in programming
  - Designed so that the programming practice could easily map into the development process
- Organized these into the XP onion
  - Many have these have been adopted as software engineering best practices

# Infrastructure as Code

- With the increase in hardware capability, hardware was not being used to its capacity
  - Running multiple virtual machines on a single hardware host solved the problem
  - The different VMs would talk to a hypervisor that would be responsible for allocating hardware to the virtual machine.
  - This is how a VM is created in the cloud.
- This meant that provisioning an operational environment
  - Did not mean working with hardware directly
  - But we wrote a specification or set of instructions to the hypervisor
  - The spec tells the hypervisor what virtual hardware to set up
  - The hypervisor does the hardware allocation
  - Writing this specification became called “infrastructure as code”

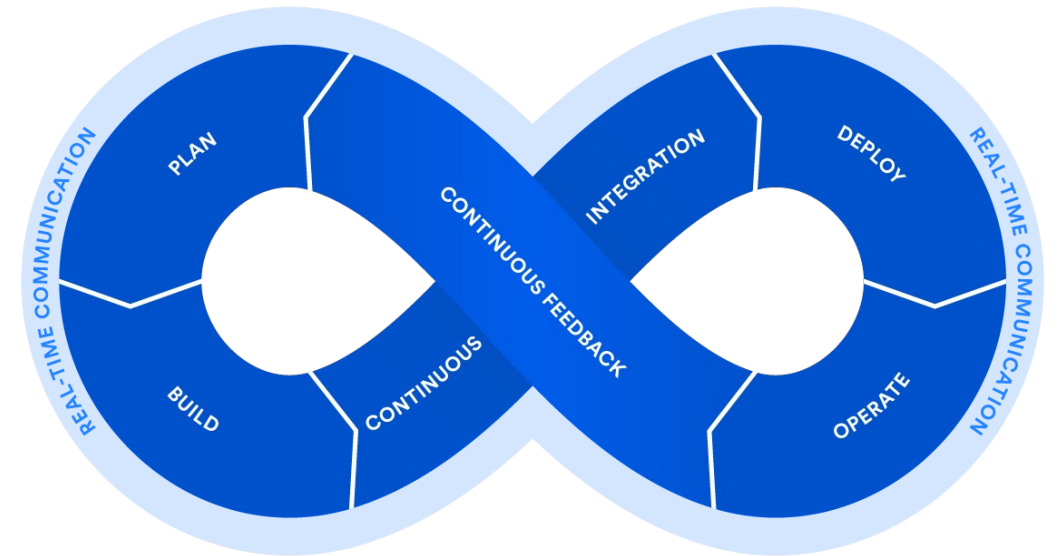
# Infrastructure as Code





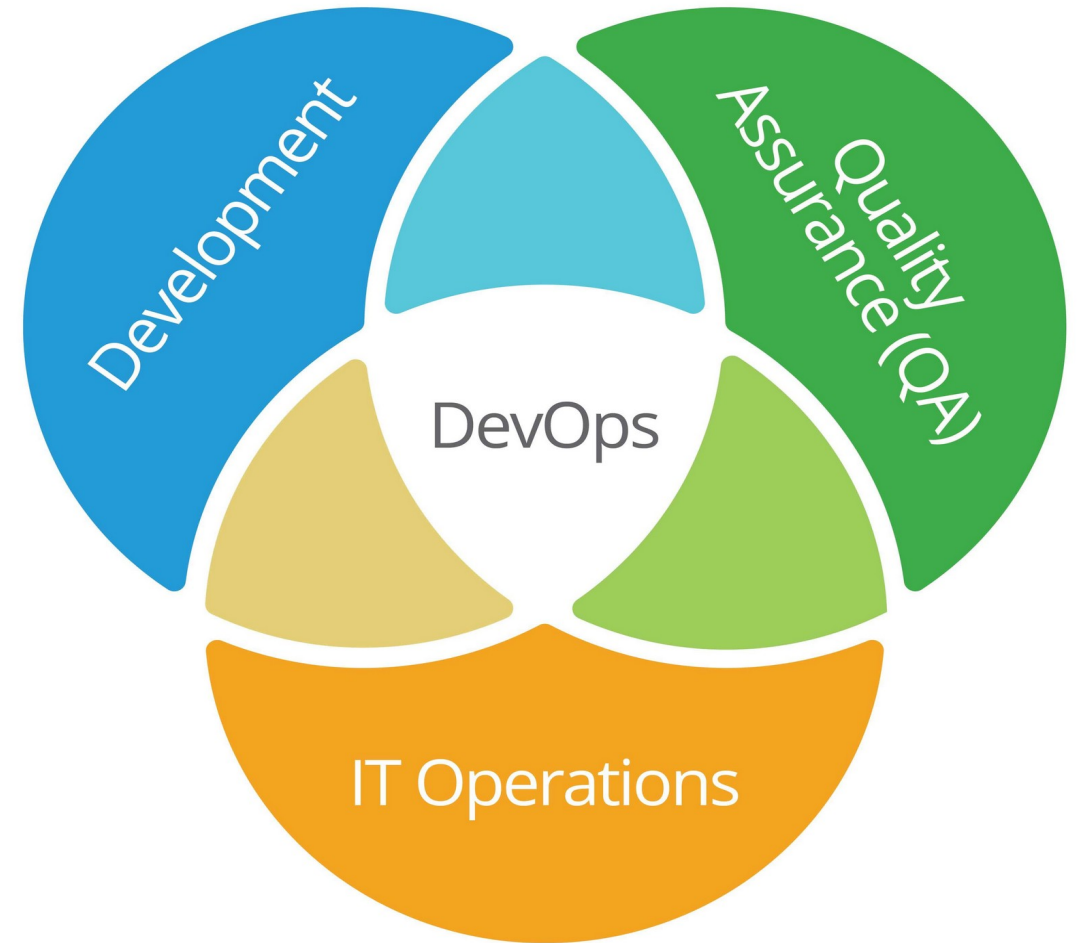
# DevOps

- Driven by virtualization and Infrastructure as code
  - Dev and Ops had been two separate worlds
  - Dev was sort of automated
  - Ops was manual and bare metal
- Virtualization turned it all into code
  - Now the same tools can be used in the entire life cycle of a software product
  - Opportunity for full process automation support
- And it allowed the integration of the engineering deployment phase to be integrated into software engineering



# The Goal of DevOps

- Desilo-ize the three areas in software development
- Get everyone using the same sorts of tools, practices and automation



# Defining CI/CD

- CI/CD is not a methodology
  - It is not Agile or DevOps
  - Although both rely on CI/CD and use it extensively
- CI/CD is process automation applied to SE
  - It is not Agile or DevOps
  - Similar to other kinds of automation
  - Improves process efficiency and effectiveness
- CI/CD is process agnostic
  - Can be used anywhere a SE process is well defined
  - Using CI/CD with bad processes makes them worse

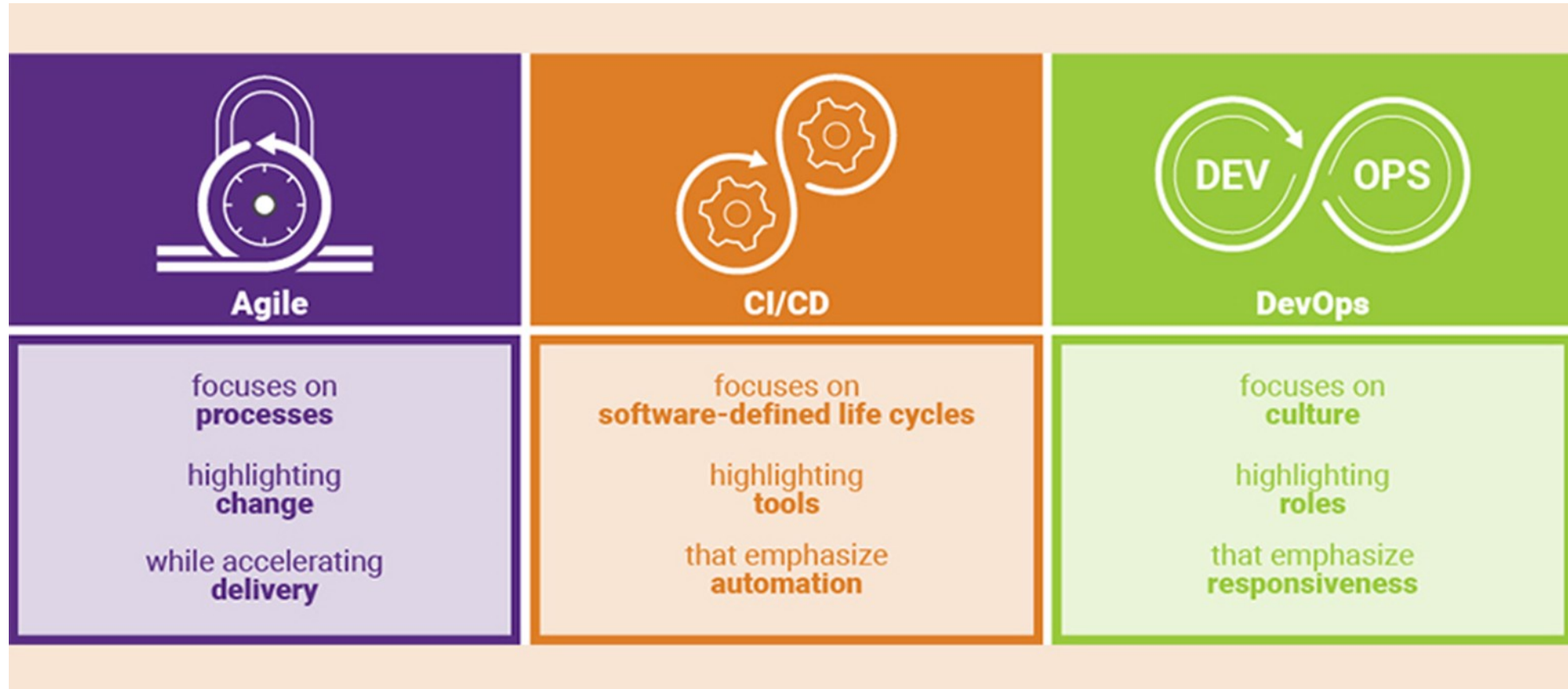
*A fool with a tool is still a fool*

Martin Fowler

*A computer lets you make more mistakes faster than any invention in human history – with the possible exceptions of handguns and tequila*

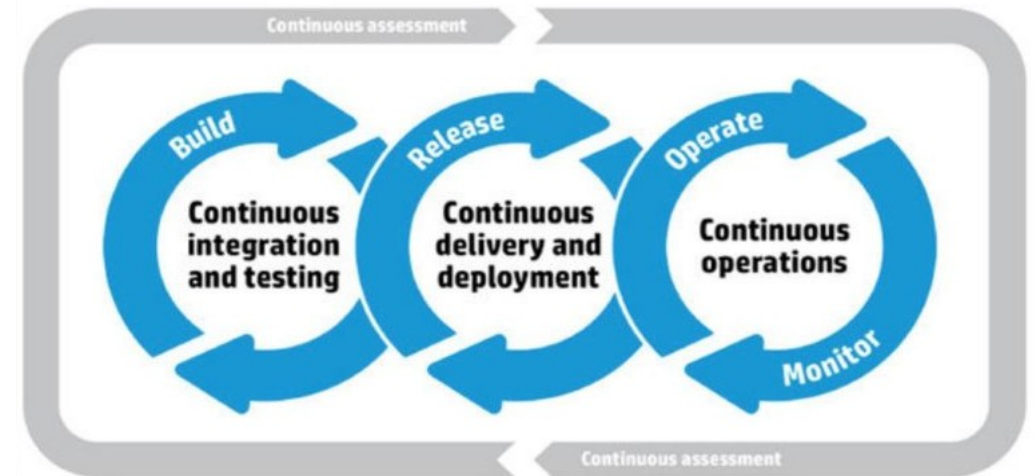
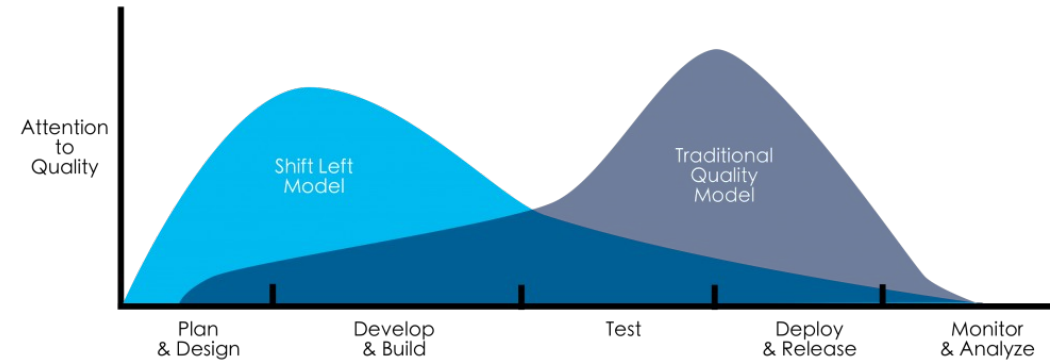
Mitch Ratcliffe

# Agile, DevOps and CI/CD



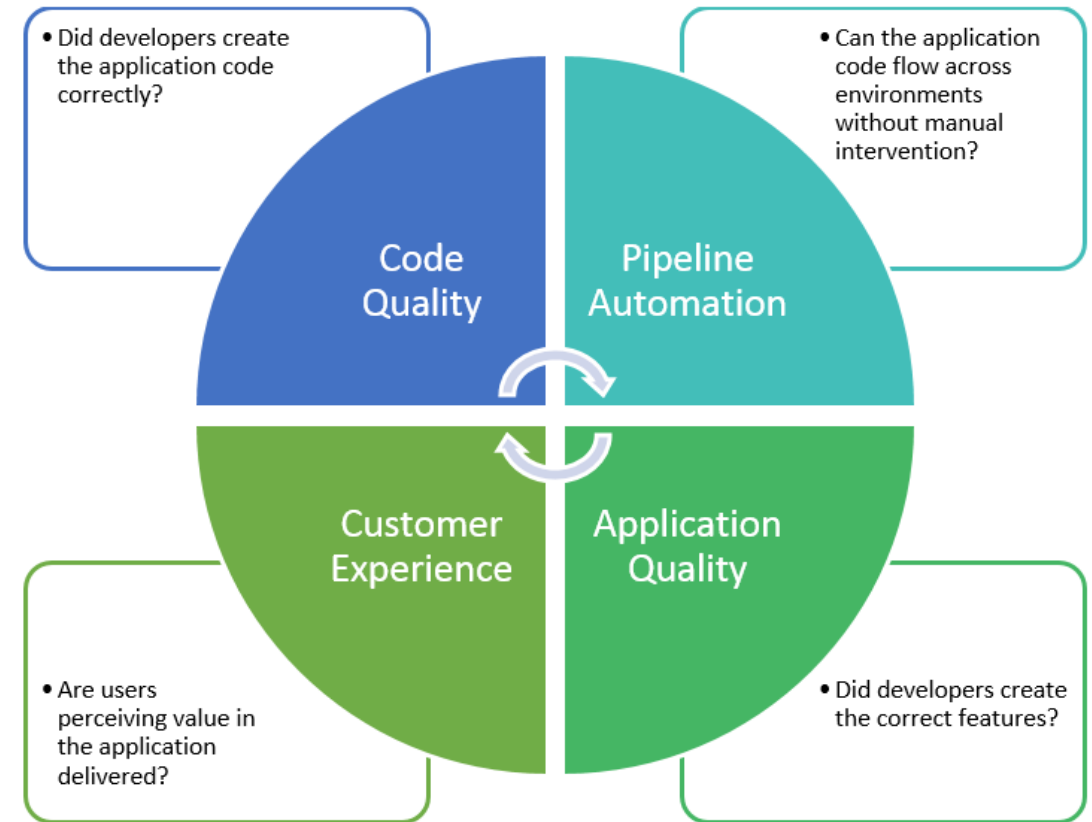
# Continuous Testing

- Continuous Testing
  - Every artifact is tested as it is created
- Shift Left Model
  - Test early, test often
- CI/CD also adds
  - Automated testing at every stage
  - CT is triggered by events in the CI/CD process
  - Checking in code => automated unit testing
  - Build => integration testing

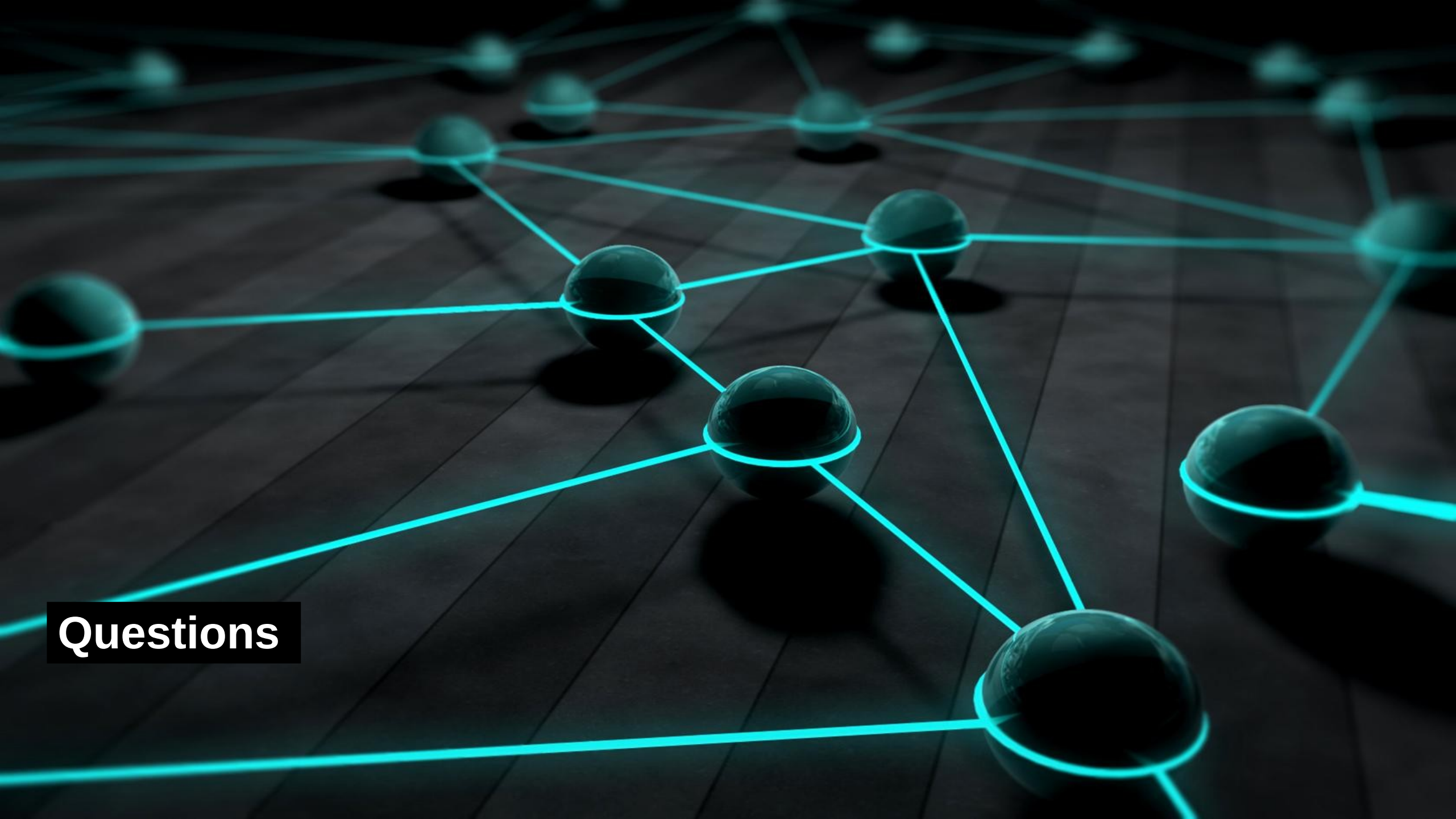


# Continuous Testing

- Does not replace human based testing
  - Like pair programming and code reviews
  - Creates “quality gates”
- Development pipelines abort when tests fail
  - Adding continuous security testing and security planning is called DevSecOps







**Questions**